A Scalable Framework for Automatic Playlist Continuation on Music Streaming Services

Walid Bendada
Deezer Research
LAMSADE, Université Paris Dauphine, PSL
research@deezer.com

Guillaume Salha-Galvan
Deezer Research

Thomas Bouabça
Deezer Research

Tristan Cazenave
LAMSADE, Université Paris Dauphine, PSL

ABSTRACT

Music streaming services often aim to recommend songs for users to extend the playlists they have created on these services. However, extending playlists while preserving their musical characteristics and matching user preferences remains a challenging task, commonly referred to as *Automatic Playlist Continuation* (APC). Besides, while these services often need to select the best songs to recommend in real-time and among large catalogs with millions of candidates, recent research on APC mainly focused on models with few scalability guarantees and evaluated on relatively small datasets. In this paper, we introduce a general framework to build scalable yet effective APC models for large-scale applications. Based on a represent-then-aggregate strategy, it ensures scalability by design while remaining flexible enough to incorporate a wide range of representation learning and sequence modeling techniques, e.g., based on Transformers. We demonstrate the relevance of this framework through in-depth experimental validation on Spotify's Million Playlist Dataset (MPD), the largest public dataset for APC. We also describe how, in 2022, we successfully leveraged this framework to improve APC in production on Deezer. We report results from a large-scale online A/B test on this service, emphasizing the practical impact of our approach in such a real-world application.

CCS CONCEPTS

• **Information systems** → *Recommender systems*; *Personalization*.

KEYWORDS

Automatic Playlist Continuation, Scalability, Music Recommender Systems, Music Streaming Services, A/B Testing.

ACM Reference Format:
Walid Bendada, Guillaume Salha-Galvan, Thomas Bouabça, and Tristan Cazenave. 2023. A Scalable Framework for Automatic Playlist Continuation on Music Streaming Services. In *Proceedings of the 46th International ACM SIGIR Conference on Research and Development in Information Retrieval (SIGIR '23), July 23–27, 2023, Taipei, Taiwan.* ACM, New York, NY, USA, 11 pages. https://doi.org/10.1145/3539618.3591628

SIGIR '23, July 23–27, 2023, Taipei, Taiwan

ACM ISBN 978-1-4503-9408-6/23/07.
https://doi.org/10.1145/3539618.3591628

1 INTRODUCTION

Recommender systems are becoming increasingly important for music streaming services such as Apple Music, Deezer, or Spotify [6, 24, 46]. While these services provide access to ever-growing musical catalogs, their recommender systems prevent information overload problems by identifying the most relevant content to showcase to each user [3, 45]. Recommender systems also enable users to discover new songs, albums, or artists they may like [24, 46]. Overall, they are widely regarded as effective tools to improve the user experience and engagement on these services [3, 5, 38, 62].

In particular, music streaming services often recommend songs for users to continue the personal *playlists* they have created on these services. Broadly defined as ordered sequences of songs intended to be listened to together, playlists often comply with specific music genres, moods, cultural themes, or activities [2, 5, 60]. Automatically extending them while preserving their musical characteristics and matching user preferences remains a challenging task, commonly referred to as *Automatic Playlist Continuation* (APC) [7, 46, 54, 60]. Although there have been attempts to tackle APC predating music streaming services (see, e.g., the survey of Bonnin et al. [4]), the advent of these services has driven music listeners towards more and more playlist consumption [25], and the need for viable APC solutions has been growing ever since. APC is now considered one of the "*most pressing current challenges in music recommender systems research*" by Schedl et al. [46].

As APC is closely related to sequence-based and session-based recommendation [8, 56], recent research on this problem extensively focused on leveraging models already successful in other sequence modeling tasks, notably language modeling, for recommendation purposes. Drawing on the analogy that exists between words within sentences on the one hand, and songs within playlists on the other hand, researchers have proposed various effective APC models based on word2vec [52, 55], recurrent neural networks [10, 27], convolutional neural networks [59], and attention mechanisms [19]. However, as we will detail in Section 2, these promising studies often overlooked scalability concerns and evaluated their models on relatively small subsets of real-world datasets.

Yet, scalability is essential for industrial applications on music streaming services, which must select the best songs to recommend for APC among large catalogs with tens of millions of candidates. Besides the number of songs to handle when training the APC model, scalability of inference, while being sometimes neglected, is also crucial. As we will further elaborate in Sections 3 to 5, music streaming services must usually perform APC online, repeatedly,

and in real-time. Indeed, users regularly create new playlists and modify the existing ones. In return, they expect the system to provide updated recommendations, with minimal delay.

In summary, there is a discrepancy between the complexity and evaluation procedure of APC models proposed in the recent scientific literature, and the scalability requirements associated with industrial applications. As an illustration of this discrepancy, we stress that modern sequence modeling techniques were seldom used during the RecSys 2018 APC Challenge [26, 60]. Aiming to foster research on large-scale APC, this open challenge was evaluated on Spotify's Million Playlist Dataset (MPD), which, to this day, remains the largest public dataset for APC [7]. To provide APC recommendations on this dataset, many teams favored relatively simpler methods during the challenge (see our review in Section 2) and, as of today, leveraging the modern APC models discussed above in such a large-scale setting still poses scalability challenges.

In this paper, we propose a general framework to overcome these challenges and remedy this observed discrepancy. While remaining versatile enough to incorporate a wide range of modern methods, our solution formally characterizes the requirements expected from suitable APC solutions for large-scale industrial applications. More precisely, our contributions in this paper are listed as follows:

- We introduce a principled framework to build scalable yet effective APC models. Based on a *represent-then-aggregate* strategy, it permits incorporating a wide range of complex APC models into large-scale systems suitable for industrial applications. Our framework systematically decomposes these models into a part handling song representation learning, and a part dedicated to playlist-level sequence modeling.
- We illustrate the possibilities induced by our framework, showing how one can design scalable APC models benefiting from the most popular architectures for sequence modeling, e.g., recurrent neural networks [22] or Transformers [53], combined with complex song representation learning models, e.g., neural architectures processing metadata [40].
- We demonstrate the empirical relevance of our framework for large-scale APC through in-depth experimental validation on Spotify's MPD dataset, the largest public APC dataset. Along with this paper, we publicly release our source code on GitHub to ensure the reproducibility of our results.
- We also describe how, in 2022, we leveraged this framework to improve APC in production on the global music streaming service Deezer. We present results from a large-scale online A/B test on users from this service, emphasizing the practical impact of our framework in such a real-world application.

This article is organized as follows. In Section 2, we introduce the APC problem more precisely and review previous work. In Section 3, we present our proposed framework to build scalable yet effective APC models, providing a detailed overview of its possibilities and limitations. We report our experimental setting and results on the public MPD dataset in Section 4, and describe our online A/B test on the Deezer service in Section 5. Finally, we conclude in Section 6.

2 PRELIMINARIES

We begin this section by precisely defining the APC problem we aim to solve. We subsequently review the relevant related work.

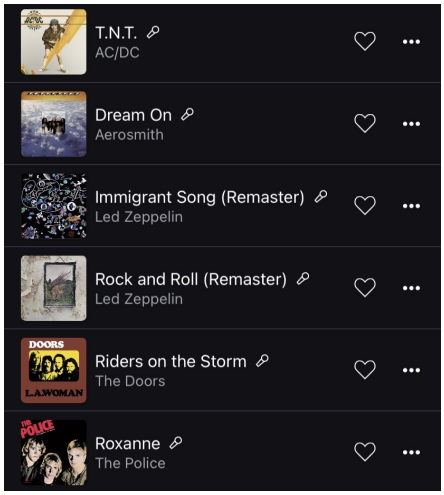


Figure 1: An example of a user playlist on Deezer.

2.1 Problem Formulation

2.1.1 Notation. This paper considers a catalog $\mathcal{S} = \{s_1, ..., s_N\}$ of $N \in \mathbb{N}^*$ songs available on a music streaming service. Users from this service can create playlists containing these songs, as in Figure 1. They can update their existing playlists at any time, by adding, removing, or reordering songs. We denote by $L \in \mathbb{N}^*$ the maximum length for a playlist, a parameter fixed by the service. We denote by $\mathcal{P}$ the set of all playlists that can be created from $\mathcal{S}$:

$$\mathcal{P} = \bigcup_{l=1}^{L} \mathcal{S}^l. \tag{1}$$

Lastly, we associate each song $s \in \mathcal{S}$ with a descriptive tuple m_s of size $M \in \mathbb{N}^*$. This tuple captures metadata information on the song, e.g., the name of the artist or the album it belongs to:

$$m_s \in \mathcal{M} = \mathcal{M}_1 \times \mathcal{M}_2 \times \cdots \times \mathcal{M}_M. \tag{2}$$

In this equation, each $\mathcal{M}_i$ denotes the set of possible values for a given metadata type, e.g., the set of possible artists.

2.1.2 APC. In such a setting, the APC problem consists in extending a playlist $p \in \mathcal{P}$ by adding more songs matching the target characteristics of p [7, 46]. Formally, an APC model is a function:

$$f : \mathcal{P} \times \mathcal{S} \to \mathbb{R}, \tag{3}$$

associating each $p \in \mathcal{P}$ and $s \in \mathcal{S}$ with a "relevance" or "similarity" score $f(p, s)$. The higher the score, the more likely s will fit well as a continuation of p, according to f. To assess and compare the performance of APC models, the standard methodology consists in:

- Collecting a set of test playlists, unseen during training.
- Then, masking the last songs of these test playlists.
- Finally, evaluating the ability of each model to complete these partially masked sequences using song relevance scores, computing metrics such as the ones we will consider in Section 4.

2.2 Related Work

In recent years, the APC problem has garnered significant attention from researchers, leading to substantial efforts to address it.

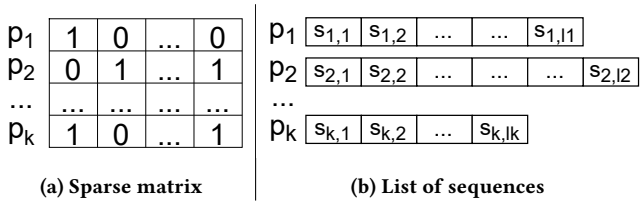
2.2.1 Collaborative Filtering for APC. Historically, Collaborative Filtering (CF) [30, 49] has been a prevalent approach for APC [11, 34, 46, 54]. As illustrated in Figure 2a, CF methods for APC usually represent playlist datasets as sparse matrices where each row corresponds to a playlist, each column corresponds to a song, and the associated binary value indicates whether the song appears in the playlist. In essence, they aim to infer the relevance of a song within a playlist by leveraging the content of similar playlists. Some of these CF methods compute inner products between pairs of playlists sparse vectors on one side, pairs of songs sparse vectors on the other side, and use the information of song occurrence within playlists to compute similarities between playlists and songs [11, 54]. Others assume that the sparse playlist-song occurrence matrix has a low-rank structure and depict it as the product of two rectangular dense matrices using Matrix Factorization (MF) techniques [31, 34]. The first matrix, of dimension $K \times D$ (with K the number of playlists in the dataset, and some fixed $D \ll \min(K, N)$), represents a playlist by row. The second one, of dimension $D \times N$, represents a song by column. They can be interpreted as D-dimensional vectorial representations of playlists and songs in a common "embedding" vector space. In this space, one can compute the similarity between any pair, e.g., using inner products [23]. As these CF methods usually fail to provide recommendations for songs absent from existing playlists [18, 40], researchers have also proposed to integrate metadata information into predictions, with the underlying assumption that co-occurrence information of frequent songs should flow to rare songs sharing similar attributes. Notable examples of CF models integrating metadata include Factorization Machines (FM) [40]. FM represent metadata in the same embedding space as playlists and songs, and add inner products between embedding vectors of various metadata to the original prediction score. They have been successfully used for APC [13, 34, 43, 58].

2.2.2 Sequence Modeling for APC. Over the past few years, an increasing number of research works on APC have proposed to represent playlist datasets, not as sparse interaction matrices, but as uneven lists of song sequences, as illustrated in Figure 2b. These studies have adapted techniques already successful in other sequence modeling tasks, notably language modeling, to extend the ordered sequences of songs that playlists naturally constitute [10, 19, 52, 55, 59]. More specifically, they have introduced APC models based on word2vec [52, 55], recurrent neural networks [10, 27], convolutional neural networks [59], and attention mechanisms [19]. In the experimental evaluations of these studies, such sequential methods often outperform the pure CF approaches from Section 2.2.1. However, these promising results were obtained on relatively small subsets[1] of real-world datasets (see, e.g., the first four lines of Table 1), due to the absence of larger public real-world datasets (for studies predating the MPD release [7]) but also to intrinsic scalability issues related to some of these models [60]. Hence, their practical effectiveness in large-scale applications involving millions of songs and playlists still needed to be fully demonstrated. Regarding the most recent class of sequence modeling neural architectures, i.e., Transformers [53], they have been proposed for sequential product recommendation (see, e.g., SASRec and BERT4Rec [28, 50]), but we have not seen examples of usage on large-scale APC. Similarly, nearest neighbors techniques have been proposed for sequence prediction, sometimes outperforming complex neural models [33, 35, 36], but were not evaluated on large-scale APC. Finally, while some approaches leverage the multi-modal aspect of APC and others focus on its sequential nature, to our knowledge, few attempts have been made to explicitly consider both components while remaining efficient enough for large-scale applications.

[1]In particular, the range of recommendable songs was often restricted to the most popular ones. Besides scalability concerns, such a restriction also questions the ability of these models to recommend songs from the *long tail*, a matter of great importance considering that the size of the catalog is one of the specificities of music recommendation [46] and that numerous models tend to be biased towards popular artists [32].

Table 1: Public datasets for APC.

Dataset	Songs	Playlists	Interactions
AOTM [37]	91 166	27 005	306 830
NOWPLAYING [61]	75 169	75 169	271 177
30MUSIC [51]	210 633	37 333	638 933
MELON [14]	649 091	148 826	5 904 718
MPD [7]	2 262 292	1 000 000	66 346 428



(a) Sparse matrix **(b) List of sequences**

Figure 2: An APC dataset represented as (a) a sparse playlist-song occurrence matrix, (b) a list of uneven song sequences.

2.2.3 Towards Large-Scale APC. To foster research on large-scale APC, the 12th ACM Conference on Recommender Systems hosted the "RecSys 2018 APC Challenge" [7], evaluated on the Million Playlist Dataset (MPD), which was then publicly released. Composed of one million playlists created by Spotify users, the MPD stands out for its magnitude compared with other public datasets from Table 1, as well as for its sparsity. Indeed, while more than two million songs can be found in the dataset, roughly 60% of them do not appear more than twice within playlists [7]. Spotify also provided side information on songs and playlists. In practice, such information often helps overcome the cold start problem induced by data sparsity [6, 44]. In 2019, Zamani et al. [60] analyzed key insights from the challenge, explaining that many teams leveraged:

- Ensemble architectures: several teams combined several APC models of the form $f : \mathcal{P} \times \mathcal{S} \to \mathbb{R}$, each of them providing different "candidate" songs to extend playlists.
- Or, two-stage architectures: a first simple model rapidly scored all songs from $\mathcal{S}$, retrieving a subset of "candidate" songs, several orders of magnitude smaller than the original song set. Then, some more sophisticated model(s) re-ranked candidate songs to improve APC recommendations. The performance of such two-stage strategies was highly dependent on the quality of candidate retrieval models [60].

Overall, the modern sequence modeling neural networks from Section 2.2.2 were seldom used for candidate retrieval in these systems,

either because they were used in a way that did not scale to millions of songs, or led to underperforming results with respect to alternatives [60]. When considering both candidate selection models of ensemble/two-stage architectures, as well as single-stage architectures directly scoring all songs, most approaches leveraged simpler and faster non-parametric nearest neighbor models, applied to various CF-based song/playlist embedding representations learned using models from Section 2.2.1. In summary, most teams favored simpler and more scalable methods than the ones presented in Section 2.2.2 despite their promising performances on smaller datasets. We believe this challenge highlighted the current discrepancy between the complexity and evaluation procedure of APC models proposed in the recent scientific literature, and the simpler ones successfully used for large-scale applications. In particular, in the context of a global questioning of the ability of deep learning to actually improve recommender systems [12, 36, 41], we believe that research on APC would benefit from a formal characterization of the requirements expected from suitable solutions for large-scale APC applications, which we provide in the remainder of this paper.

3 LARGE-SCALE PLAYLIST CONTINUATION

In this section, we present and analyze our proposed framework to build scalable yet effective APC models for large-scale applications.

3.1 Objectives and Requirements

Our main goal is to provide a comprehensive set of guidelines for implementing APC models that are scalable by design. Additionally, we aim to maintain flexibility and generality in our approach.

Specifically, in this paper, our definition of *scalability* is twofold. Firstly, we want to build models that can handle large datasets with millions of songs and users, as is often required in real-world applications on music streaming services. Furthermore, insights from industrial practitioners have emphasized that scalability is also essential during the *inference* phase [41]. In practice, while regularly training an APC system is necessary to incorporate new data (e.g., get information on new songs and playlists, and consequently update model parameters), these updates can be purposely delayed until a batch of new events has been observed. This allows model training operations to be performed *offline on a regular schedule*, such as once per day, and enables the model to take more time to update its parameters [42]. In contrast, the inference process, repeatedly making recommendations to users, must be accomplished:

- *Online* on the service: as users regularly create and update playlists, the exact song sequence that must be extended by the APC model can *not* be processed offline in advance.
- With minimal delay: to minimize latency costs when providing updated recommendations to users. For example, on a global service like Deezer, an inference time longer than *a few tens of milliseconds* would be unacceptable in production.

For these reasons, scalability of inference will be particularly crucial in our framework. Consequently, we will aim to minimize the number of *online* operations whose complexity depends on the size of the dataset, described by N, L, M, and $\max_{1\leq k\leq M}|\mathcal{M}_k|$. In practice, the number of songs N is the largest of these parameters. Therefore, we will prioritize the *offline* pre-computation of operations depending on N, whenever possible.

3.2 A Scalable Framework for APC

We now formally introduce our framework for APC at scale. We focus[2] on single-stage models $f\colon \mathcal{P}\times\mathcal{S}\rightarrow\mathbb{R}$ directly scoring each song $s\in\mathcal{S}$ with a similarity score $f(p,s)$ for extending some playlist $p\in\mathcal{P}$. Also, we focus[2] on APC models learning "embedding" vectorial representations for any playlist p and song s. We denote them by $\mathbf{h}_p\in\mathbb{R}^D$ and $\mathbf{h}_s\in\mathbb{R}^D$, respectively, for some embedding dimension $D\in\mathbb{N}^*$. Such models can be written as follows:

$$f(p,s)=\text{SIM}(\mathbf{h}_p,\mathbf{h}_s), \tag{4}$$

for some similarity function $\text{SIM}\colon\mathbb{R}^D\times\mathbb{R}^D\rightarrow\mathbb{R}$. For each element of Equation (4), we establish properties that must be verified in our framework, following the principles from Section 3.1.

3.2.1 Song Representation. Firstly, we note that each song embedding vector $\mathbf{h}_s$ depends on the song $s\in\mathcal{S}$ and, possibly, the metadata $m_s\in\mathcal{M}$. As an embedding vector must be computed for each of the N songs of the catalog, we require to perform this operation *offline in advance*, with its result stored to be rapidly accessible with a simple read operation. Consequently, song embedding vectors should *not* depend on the playlist p to extend. Formally, in our framework, each $\mathbf{h}_s$ vector should be computed via a function[3]:

$$\phi\colon\mathcal{S}\rightarrow\mathbb{R}^D, \tag{5}$$

learning a unique vectorial representation for each song.

3.2.2 Playlist Representation. On the other hand, the representation $\mathbf{h}_p$ of each playlist $p\in\mathcal{P}$ can not be computed offline in advance, as playlists can be created and updated at any time by users (see Sections 2.1 and 3.1). Hence, according to our requirements, the playlist-level representation learning operations should not depend on a large set of inputs. We propose to learn $\mathbf{h}_p$ solely from the song sequence $(s_{p1},...,s_{pl})$ characterizing p. Since large-scale APC usually implies dealing with sparse data [7], our framework also aims to benefit from parameter sharing whenever possible, by adding two constraints on the function learning $\mathbf{h}_p$:

- It should leverage song representations $\mathbf{h}_s$ as input.
- It should be independent of the playlist length $l\in\{1,\ldots,L\}$.

Hence, by defining $\mathcal{E}=\bigcup_{l=1}^{L}\mathbb{R}^{D\times l}$, each $\mathbf{h}_p$ should be computed via:

$$g\colon\mathcal{E}\rightarrow\mathbb{R}^D, \tag{6}$$

a function *aggregating* representations from the songs present in the playlist p into a single playlist embedding representation $\mathbf{h}_p$.

3.2.3 Similarity. The final scoring operation $f(p,s)=\text{SIM}(\mathbf{h}_p,\mathbf{h}_s)$, estimating how likely each song s will fit well as a continuation of some playlist p using the above $\mathbf{h}_s$ and $\mathbf{h}_p$, constitutes the complexity bottleneck of the APC prediction process. Indeed, it depends on N, assuming that the APC model actually considers all songs from $\mathcal{S}$ when making recommendations (and not some approximate subset). Moreover, it can not be pre-computed offline beforehand, as it depends on the playlist representation, which is itself computed online when we model observes the exact song sequence to

[2]This focus is made without loss of generality. Indeed, single-stage APC models stemming from our scalable framework could be integrated into two-stage/ensemble systems such as the ones from Section 2.2.3. Also, while embedding representations are ubiquitous in the related work from Sections 2.2.1 and 2.2.2, our framework could be adapted to settings where $\mathbf{h}_p$ and $\mathbf{h}_s$ capture more general descriptive information.
[3]We imply the metadata in dependencies, as each m_s exclusively depends on s.

extend. Although similarity functions based on neural networks have been proposed [21], Rendle et al. [41] have demonstrated that they do not outperform the traditional inner product, whose ability to rapidly score millions of items makes it a natural choice for use as a similarity function in our framework: $f(p, s) = \langle \mathbf{h}_p, \mathbf{h}_s \rangle$.

3.2.4 Represent-Then-Aggregate. In summary, as illustrated in Figure 3, our framework constraints APC models to adopt the structure:

$$f(p, s) = \langle g(\mathbf{h}_{s_{p1}}, ..., \mathbf{h}_{s_{pl}}), \mathbf{h}_s \rangle, \text{ with } \forall k \in \mathcal{S}, \mathbf{h}_k = \phi(k). \quad (7)$$

As it involves a function ϕ representing songs and a function g aggregating these representations, we refer to our framework as *Represent-Then-Aggregate* (RTA) in the following. Our experiments from Sections 4 and 5 will empirically confirm the scalability of various APC models complying with this framework.

3.3 Examples of RTA Architectures

While ensuring scalability by design, our RTA framework remains flexible enough to incorporate a wide range of modern models. In particular, our experiments will consider the following options for song representation learning and playlist-level aggregation.

3.3.1 Song Representation Function ϕ.

- $\mathbf{h}_s = \phi_e(s) = \mathbf{e}_s$: we directly associate each song s with a vector $\mathbf{e}_s \in \mathbb{R}^D$ using a representation learning model trained from playlist-song occurrence data (see Section 4.2.1).
- $\mathbf{h}_s = \phi_{\text{FM}}(s) = \sum_{m \in m_s} \mathbf{e}_m$: we represent each song by the sum or average of embedding vectors associated with its metadata, which we themselves learn using a model trained from occurrence data. We refer to this approach as FM due to its connection to Factorization Machines from Section 2.2.1.
- $\mathbf{h}_s = \phi_{\text{NN}}(s) = \text{NN}(\{\mathbf{e}_m, \forall m \in m_s\})$: we represent each song by the output of a neural network NN processing song metadata. Our experiments will consider an attention-based neural architecture similar to the one of Song et al. [47].

3.3.2 Playlist Sequence Modeling / Aggregation Function g.

- $g_{\text{AVG}}(\mathbf{h}_{s_{p_1}}, ..., \mathbf{h}_{s_{p_l}}) = \frac{1}{l} \sum_{i=1}^{l} \mathbf{h}_{s_{p_i}}$: the playlist representation is the average representation of the songs it contains.
- $g_{\text{CNN}}(\mathbf{h}_{s_{p_1}}, ..., \mathbf{h}_{s_{p_l}}) = \text{CNN}([\mathbf{h}_{s_{p_1}}; ...; \mathbf{h}_{s_{p_l}}])$: a gated convolutional neural network (CNN) processing a concatenation of song representations learns the playlist representation. A key component of the ensemble model that won the RecSys 2018 APC challenge [54], the use of such gated CNN for APC was inspired by its successful use in language modeling [9].
- $g_{\text{GRU}}(\mathbf{h}_{s_{p_1}}, ..., \mathbf{h}_{s_{p_l}}) = \text{GRU}([\mathbf{h}_{s_{p_1}}; ...; \mathbf{h}_{s_{p_l}}])$: a recurrent neural network (RNN) composed of gated recurrent units (GRU) learns the playlist representation. Our experiments will consider an architecture similar to the one of Hidasi et al [22] but with a different loss, presented in Section 3.4.
- $g_{\text{Transformer}}(\mathbf{h}_{s_{p_1}}, ..., \mathbf{h}_{s_{p_l}}) = \text{Decoder}([\mathbf{h}_{s_{p_1}}; ...; \mathbf{h}_{s_{p_l}}])$: the Decoder part of a Transformer network [53] learns the playlist representation. The choice of keeping only the Decoder for APC was motivated by the recent surge of GPT models, which have demonstrated state-of-the-art performances on the analogous task of sentence continuation [39].

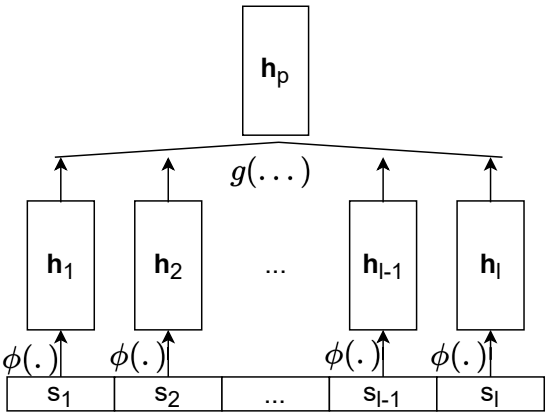


Figure 3: Our *Represent-Then-Aggregate* framework for APC.

3.3.3 Limitations. These examples illustrate the versatility of our RTA framework. Simultaneously, we are also aware of some limitations it imposes. In particular, Equation (7) can not express APC architectures inspired by word2vec models [52, 55]. Indeed, they leverage *different* song representations depending on whether they are used to characterize the context of other songs, or evaluated as candidates for APC. Leveraging different representations is relevant for language modeling, since words within a sentence are rarely synonyms and should not be given close representations, yet models attempt to represent words by their context, motivating the need for different representations for each word (see the discussion by Goldberg et al. [16]). On the contrary, in the case of APC, we believe songs appearing in the same playlist should have close representations, i.e., songs should be similar to their context (an assumption at the core of matrix factorization techniques [31]). For this reason, our framework excludes these approaches in favor of the scalability induced by more parameter sharing.

3.4 Example of a Training Procedure

While our RTA framework remains general, we suggest the following training procedure, which can be applied to optimize all models stemming from this framework. Our experiments in this paper will adopt this same procedure to train all RTA-based APC models.

Our preliminary experiments have shown that leveraging pretrained embedding vectors from a weighted regularized matrix factorization as initial song representations improves performances. Therefore, we propose to start by factorizingthe playlist-song occurrence matrix of the APC problem under consideration, associating each song s with some $\mathbf{e}_s \in \mathbb{R}^D$ used for initialization. Simultaneously, we initialize embedding vectors of metadata information by averaging vectors $\mathbf{e}_s$ of songs sharing the same metadata value.

Starting from these initial representations, we jointly optimize weights of the models selected as ϕ and g, using batches of playlists of varying lengths. For each playlist p of length $l \in \{2, \ldots, L\}$, we create $l - 1$ sub-playlists denoted $p_{:i}$ using the first i songs of p, with $i \in \{1, \ldots, l-1\}$. Then, we sample a song set $\mathcal{S}^-(p)$ from $\mathcal{S} \setminus p$. As we expect the APC model to return a high similarity scores for $f(p_{:i}, p_{s_{i+1}})$ and a lower score for songs that are absent from p, our procedure optimizes model weights via gradient descent

minimization [17] of the loss $\mathcal{L}(p) = \mathcal{L}_{\text{pos}}(p) + \mathcal{L}_{\text{neg}}(p)$, with:

$$\mathcal{L}_{\text{pos}}(p) = -\sum_{i=1}^{l-1} \log\Big(\frac{1}{1+e^{-f(p_{:i},p_{s_{i+1}})}}\Big), \tag{8}$$

and:

$$\mathcal{L}_{\text{neg}}(p) = -\sum_{i=1}^{l-1} \sum_{s^- \in \mathcal{S}^-(p)} \log\Big(1 - \frac{1}{1+e^{-f(p_{:i},s^-)}}\Big). \tag{9}$$

4 OFFLINE EXPERIMENTS

We now present an experimental evaluation of our framework on the Million Playlist Dataset (MPD) [7, 60]. We release our source code on GitHub[4], to ensure the reproducibility of our results and to encourage the future usage of our framework.

4.1 Experimental Setting

4.1.1 Dataset. We consider the entire MPD for our experiments. This dataset includes one million playlists created by Spotify users, using more than two million songs (see Table 1). For each song s, we have metadata information[5] corresponding to its artist (art_s), the album it belongs to (alb_s), the duration of the song (dur_s), and the number of times it occurs in the dataset, a metric referred to as popularity (pop_s). Hence, $m_s = (\text{art}_s, \text{alb}_s, \text{dur}_s, \text{pop}_s)$. As dur_s and pop_s are numerical values that can not be easily mapped to embedding vectors, we group similar values together into buckets:

- For the duration, we create linear buckets of 30 seconds each, with songs longer than 20 minutes being assigned to the same bucket. We have: $\text{bucket}_{\text{dur}}(s) = \min(40, \left\lceil \frac{\text{dur}_s}{30} \right\rceil)$.
- For the popularity, we create buckets using a logarithmic scale, so that we distinguish smaller values while higher values are assigned to the same bucket. By setting $\alpha = 45\,000$, i.e., the highest number of occurrences observed in the dataset, we have: $\text{bucket}_{\text{pop}}(s) = \min(100, 1 + 100 \times \left\lfloor \frac{\log(\text{pop}_s)}{\log(\alpha)} \right\rfloor)$.

4.1.2 Task. We consider a large-scale APC evaluation task similar to the one presented in Section 2.1.2. Specifically, we start by randomly sampling 20 000 playlists of length $l \geq 20$ songs from the MPD, to constitute a validation set and a test set of 10 000 playlists each. The remaining 980 000 playlists constitute our training set.

Then, we mask the last songs of test playlists, so that only their n_{seed} first songs are visible. We consider ten different configurations, with n_{seed} varying from 1 to 10, randomly selecting 1 000 test playlists for each configuration. Our experiments consist in assessing the ability of several APC models trained on the 980 000 train playlists (see Section 4.2) to retrieve the masked songs of each test playlist. Consistently with the RecSys 2018 APC Challenge, we require each APC model to predict a ranked list of $n_{\text{reco}} = 500$ candidate songs to continue each test playlist.

4.1.3 Metrics. In the following, we analyze nine different metrics to evaluate each model on the specified APC task. Firstly, we consider five *accuracy-oriented* metrics. Besides the prevalent Precision and Recall scores [46], we report the three metrics used for evaluation during the RecSys 2018 APC Challenge [7]:

- Normalized Discounted Cumulative Gain (NDCG): acts as a measure of ranking quality. It increases when the ground truth masked songs are placed higher in the ranked list of candidate songs recommended by the APC model [57].
- R-Precision: jointly captures the proportion of masked songs and artists recommended by the model. Artist matches increase the score even if the predicted song is incorrect [7].
- Clicks: indicates how many batches of ten candidate songs must be recommended (starting with the top ten candidates) before encountering the first ground truth song. Unlike previous metrics, Clicks should, therefore, be minimized [7].

Zamani et al. [60] provide exact formulas for these three metrics. In addition, we compute two *popularity-oriented* scores, described by Ludewig and Jannach [33]. They monitor the tendency of each model to cover the entire catalog when providing recommendations:

- Coverage: computes the percentage of songs from the catalog recommended at least once during the test phase.
- Popularity bias: averages the popularity of recommended songs, computed by counting the occurrences of each song in the training set and applying min-max normalization [33]. Low scores indicate that less popular songs are recommended.

Finally, to compare the *scalability* of each model, we compute:

- Training time: the time required for a model to learn from the training set until no more improvement on the validation set could be observed, regarding any accuracy-oriented metrics.
- Inference time: the average time required to recommend a list of $n_{\text{reco}} = 500$ candidate songs to extend a test playlist.

4.2 APC Models

We compare the scalability and performance of ten APC models.

4.2.1 Models based on our RTA framework. Firstly, we consider six different RTA models, built by leveraging the following song representation and playlist aggregation/modeling functions:

- MF-AVG[6]: employs the representation function $\phi_{e-\text{MF}}$ (see below), combined with the aggregation function g_{AVG}.
- MF-CNN[6]: $\phi_{e-\text{MF}}$ combined with g_{CNN}.
- MF-GRU: $\phi_{e-\text{MF}}$ combined with g_{GRU}.
- MF-Transformer: $\phi_{e-\text{MF}}$ combined with $g_{\text{Transformer}}$.
- FM-Transformer: ϕ_{FM} combined with $g_{\text{Transformer}}$.
- NN-Transformer: ϕ_{NN} combined with $g_{\text{Transformer}}$.

In the above list, we use the notation from Section 3.3. We additionally denote by ϕ_{e-MF} a particular example of representation learning function ϕ_e. Specifically, ϕ_{e-MF} initializes song embedding vectors using the weighted regularized matrix factorization mentioned in Section 3.4 and directly refines these representations via the minimization of the loss defined in this same section. All six models learn embedding vectors of dimension $D = 128$. We optimized them via the procedure of Section 3.4, using stochastic

[4] https://github.com/deezer/APC-RTA

[5] The MPD also includes the *title* of each playlist as an additional information [7]. However, Zamani et al [60] have reported that this information did not significantly improve performances during the RecSys 2018 APC Challenge. Moreover, titles are often unavailable for APC, as well as for closely related tasks such as radio generation/personalization [5]. For these reasons, we omit playlist titles in these experiments.

[6] The ensemble system that won the RecSys 2018 Challenge included such a model [54].

Table 2: Automatic Playlist Continuation (APC) on the Million Playlist Dataset (MPD) [7], using various models stemming from our proposed Represent-Then-Aggregate (RTA) framework and other baselines. Scores are computed on test playlists and averaged for n_{seed} varying from 1 to 10. All models are required to provide a ranked list of $n_{\text{reco}} = 500$ candidate songs to continue each test playlist. All embedding models learn representations of dimension $D = 128$, with other hyperparameters set as described in Section 4.2. The two columns *"Acceptable for MSS?"* indicate whether the reported training and inference times would be acceptable (✓) or not (×) for APC on a Music Streaming Service (MSS), and are discussed in Section 4.3.2.

Models	Precision (in %)	Recall (in %)	R-Precision (in %)	NDCG (in %)	Clicks (in number)	Popularity (in %)	Coverage (in %)	Total time for model training	Acceptable for MSS?	Inference time by test playlist	Acceptable for MSS?
Baselines											
SKNN	4.93 ± 0.09	36.94 ± 0.49	21.42 ± 0.30	27.66 ± 0.40	3.82 ± 0.22	14.86 ± 0.17	12.64	4 min	✓	~ 0.5 sec	×
VSKNN	4.93 ± 0.09	36.83 ± 0.49	21.27 ± 0.30	27.54 ± 0.40	3.84 ± 0.22	15.18 ± 0.17	12.28	4 min	✓	~ 0.5 sec	×
STAN	4.32 ± 0.09	32.73 ± 0.48	19.12 ± 0.28	24.26 ± 0.38	4.73 ± 0.23	13.43 ± 0.16	**27.03**	4 min	✓	~ 0.5 sec	×
VSTAN	4.59 ± 0.09	34.84 ± 0.49	20.33 ± 0.30	25.99 ± 0.40	4.49 ± 0.23	11.35 ± 0.16	20.54	3 min	✓	~ 0.5 sec	×
RTA Models											
MF-AVG	4.43 ± 0.09	32.64 ± 0.46	19.80 ± 0.28	24.46 ± 0.37	5.95 ± 0.28	21.37 ± 0.19	1.07	15 min	✓	< 0.01 sec	✓
MF-CNN	5.00 ± 0.09	37.89 ± 0.45	21.15 ± 0.26	26.83 ± 0.35	3.13 ± 0.19	15.43 ± 0.14	3.99	~ 5 h	✓	< 0.01 sec	✓
MF-GRU	5.16 ± 0.09	39.16 ± 0.46	21.83 ± 0.27	28.22 ± 0.36	2.77 ± 0.18	12.70 ± 0.13	3.18	~ 7 h	✓	< 0.01 sec	✓
MF-Transformer	5.20 ± 0.09	39.76 ± 0.47	**22.30 ± 0.28**	29.04 ± 0.38	2.60 ± 0.17	10.46 ± 0.13	5.35	~ 5 h	✓	< 0.01 sec	✓
FM-Transformer	**5.31 ± 0.09**	**40.46 ± 0.47**	22.29 ± 0.27	**29.21 ± 0.37**	2.52 ± 0.17	11.74 ± 0.13	10.18	~ 6 h	✓	< 0.01 sec	✓
NN-Transformer	5.26 ± 0.09	40.17 ± 0.47	22.18 ± 0.27	29.14 ± 0.37	**2.17 ± 0.15**	**10.33 ± 0.13**	11.81	~ 6 h	✓	< 0.01 sec	✓

gradient descent [17] with batches of 128 playlists and 100 negative samples. We tuned all hyperparameters to maximize NDCG scores on the validation set, using the *Optuna* library [1] for efficient hyperparameter search. For brevity, we report all optimal hyperparameter values in our GitHub repository[4]. All g_{CNN}, g_{GRU}, $g_{\text{Transformer}}$, and ϕ_{NN} models had from 1 to 3 layers. The width of hidden layers went from 128 to 1024 units. The kernel size for g_{CNN} models went from 2 to 5. The number of heads for $g_{\text{Transformer}}$ and ϕ_{NN} was a power of 2 ranging from 2^1 to 2^6. We tested learning rates ranging from 10^{-3} to 1, weight decays from 10^{-9} to 10^{-4}, and dropout rates from 0 to 0.5 [48]. For every model, we halved the learning rate at each epoch and performed early stopping using the validation set [17]. We used Python, training models on a single CPU machine with 25 GB of RAM and a Tesla P100 GPU accelerator.

4.2.2 Baselines. By studying these six different RTA models, our goal is to measure how well various modern methods would scale and perform using our framework, including techniques previously overlooked for large-scale APC. To establish reference points, we also concurrently evaluate and compare four non-RTA baselines:

- SKNN [20]: a session-based nearest neighbors approach that, despite its apparent simplicity, can reach competitive performances with respect to CNN and GRU models [36].
- VSKNN [33]: a variant of SKNN considering the song order within the playlist sequence as well as its popularity.
- STAN [15]: a variant of SKNN capturing the position of the song in the playlist, information from past playlists, and the position of recommendable songs in neighboring playlists.
- VSTAN [36]: combines the ideas of STAN and VSKNN into a single approach, and adds sequence-based item scoring.

We selected these four baselines for two reasons:

- Firstly, regarding performances, they tend to surpass alternatives in the recent empirical analysis of Ludewig et al. [36].
- Secondly, although they were evaluated on smaller datasets, we could train them on the MPD using our machines.

We tuned all hyperparameters by maximizing NDCG validation scores. Our set of possible values for hyperparameters is similar to Ludewig et al. [36]. These four models require timestamps of actions performed on the items of each sequence. As the MPD includes the timestamp of the latest update of each playlist and the duration of each song, we made the simplifying assumption that each song had been added right after the previous one had been listened to once.

4.2.3 Scope Limitation. For completeness, we note that we initially considered other baselines, which we ultimately discarded for scalability reasons. Most notably, we will not compare to the multimodal CF model from Yang et al. [58] and the hybrid model from Rubtsov et al. [43]. While reaching promising results during the RecSys 2018 APC Challenge, both of these two models require more than *four full days* to train on the MPD using their respective implementations (even on a machine with 28 cores and 200GB of RAM, for the latter one [43]). Considering that the MPD represents only a small fraction of playlists created by Spotify users, these models would require even more time and resources to be used by such a service. As scalability is at the core of our work, our experiments, therefore, discard these demanding baselines.

In addition, as explained in Section 3.2, we focus on single-stage APC models, directly processing and scoring all songs from $\mathcal{S}$. For clarity of exposure and comparisons, our experiments do not include systems aggregating *multiple* song scoring models. Nonetheless, it bears mentioning that the ten models from our experiments could easily be used as the first "candidate retrieving" part of a two-stage architecture (see Section 2.2.3), or as components in a larger ensemble system (e.g., Volkovs et al. [54] themselves combined MF-AVG and MF-CNN in their ensemble during the challenge). As future work, we plan to extend the scope of this paper, by examining the use of several RTA models *in conjunction* for large-scale APC.

4.3 Results

4.3.1 Performance. Table 2 reports all scores[7] on test playlists, averaged for n_{seed} varying from 1 to 10, along with 95% confidence intervals. Firstly, we confirm previous insights [36] claiming

[7]Recall: we aim to *minimize* Clicks and Popularity. We report no confidence intervals for Coverage as it considers the *total* number of recommended songs for the test set.

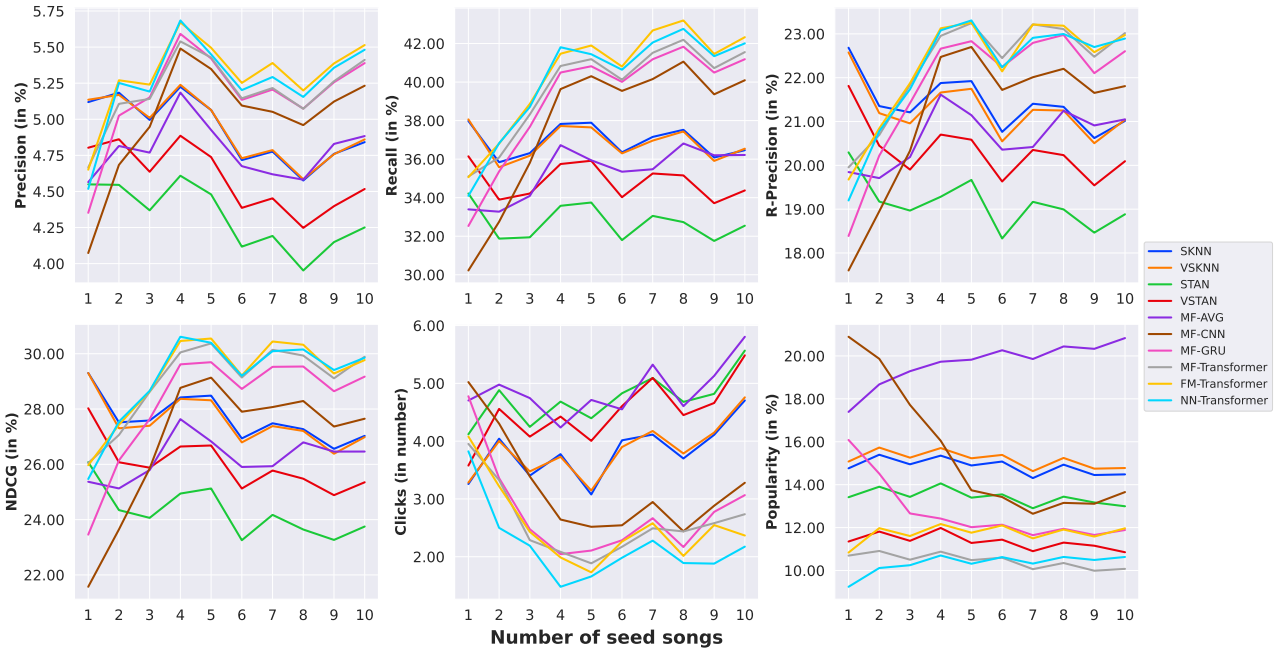


Figure 4: Automatic Playlist Continuation (APC) on the Million Playlist Dataset (MPD) [7], using the setting and models from Table 2. Contrary to this previous table, we split scores depending on n_{seed}, i.e., the number of visible songs in each test playlist.

that properly tuned nearest neighbors models, such as SKNN and VSKNN, can reach comparable performances with respect to some neural models (e.g., with a 27.66% NDCG for SKNN, vs. 26.83% for MF-CNN and 28.22% for MF-GRU). Regarding accuracy-oriented metrics, STAN, VSTAN, and MF-AVG tend to underperform, while Transformer-based models using our RTA framework achieve the best results (e.g., with a top 40.46% Recall for FM-Transformer). The FM-Transformer variant, leveraging Factorization Machine methods for song representation learning, slightly surpasses MF-Transformer and NN-Transformer on three metrics, even though all scores are statistically very close. We note that, in particular, Transformer-based models outperform alternatives in terms of ranking quality (NDCG, Clicks), a valuable property for real-world usage. This validates the relevance of our work, which specifically aims to facilitate and support the incorporation of such modern sequence modeling techniques into large-scale APC systems.

Figure 4 shows the evolution of different metrics as the number of visible songs n_{seed} in test playlists increases. While confirming previous conclusions, the figure also highlights that sequence modeling neural architectures struggle on very short playlists. For $n_{\text{seed}} \leq 2$, VSKNN and SKNN even surpass Transformers. Another interesting observation from Figure 4 and Table 2 relates to popularity-oriented scores. On average, MF-AVG provides the most mainstream recommendations (with a top 21.37% Popularity score, and a bottom 1.07% Coverage of the catalog), followed by VSKNN and SKNN. On the contrary, the good performances of Transformers do not stem from focusing only on popular songs. Interestingly, STAN and VSTAN recommend the most diverse songs (e.g., with a top 27.03% Coverage for STAN), but at the cost of a deteriorating performance.

4.3.2 Scalability. We now discuss scalability metrics. They are the ones that best showcase the relevance of our RTA framework for large-scale industrial applications. While baselines were faster to train (3 to 4 minutes on our machines), the training times associated with RTA-based models (15 minutes to a few hours) would remain acceptable for APC on a music streaming service. Indeed, as explained in Section 3.1, training operations do not have to be executed in real-time on such a service. They can be performed *offline* on a regular schedule (e.g., once per day) to update the parameters of the production model. In practice, all models evaluated in Table 2 could integrate new input data overnight.

In contrast, the *inference* process could not be delayed by a music streaming service. As detailed in Section 3.1, it would usually need to be *repeatedly* performed *online*, to provide recommendations *in real-time* to many users. In particular, we explained in this previous section that an inference time longer than a few tens of milliseconds by playlist would be unacceptable in the production environment of a service like Deezer. In Table 2, baselines require 500 milliseconds (0.5 seconds) to extend each test playlist with a list of 500 recommended songs. All RTA models perform this same operation in less than 10 milliseconds, even the most advanced models like NN-Transformer, using an attention-based network for song representation and a Transformer for playlist modeling. While all models could benefit from better hardware, overall, the inference times associated with our RTA models make them more suitable for production usage. These results highlight the ability of our framework to integrate complex but promising models, previously overlooked for large-scale APC, into effective systems meeting the scalability requirements associated with real-world industrial applications.

5 ONLINE EXPERIMENTS

This section showcases how our framework recently helped Deezer improve APC at scale. Our objective in this section is not to re-evaluate all models from Section 4, but rather to complete our analysis by further illustrating the practical relevance of our framework, through its successful application in an industrial setting.

5.1 APC on a Music Streaming Service

The global music streaming service Deezer offers an APC feature, illustrated in Figure 5. It allows millions of Deezer users to automatically extend the playlists they have created on the service. In April 2022, we conducted a large-scale online A/B test on this feature, aiming to improve recommendations by leveraging the possibilities induced by our RTA framework. Our reference system, used in production before the test and denoted "Reference" in the following, exploited a collaborative filtering model analogous to MF-AVG. It also incorporated various internal rules on the frequency of artist and album appearances in the list of recommended songs.

During the test, we instead used our RTA-based MF-Transformer, one of the three best models from Section 4, to provide APC to a randomly selected cohort of test users. We trained MF-Transformer and Reference on a private dataset of 25 million user playlists. Both models chose recommendations from a pool of two million recommendable songs. As illustrated in Figure 5, users requesting a playlist continuation were presented with a vertically scrollable list of 100 songs, with the five first ones being initially visible on mobile screens. Users were unaware of the algorithm change.

5.2 Online A/B Test Results

Firstly, our RTA framework proved to be a valuable asset in integrating MF-Transformer into Deezer's production environment. It permitted the successful integration of this model, resulting in its ability to perform APC at a minimal latency cost for users. As an illustration, we observed a 99$^{\text{th}}$ percentile inference time of only 12 milliseconds on the service, unnoticeable by users (this corresponds to the time to compute the playlist representation, score two million songs, and return 100 recommendations, using an AMD EPYC 7402P CPU machine with ten threads by inference process). This example demonstrates how our framework can enable practitioners to effectively leverage modern APC models such as Transformers, that may have otherwise been left out for scalability reasons.

Regarding performances, our test confirmed the superiority of MF-Transformer over Reference on Deezer. Using MF-Transformer improved our internal performance indicators for the APC feature, such as the Add-To-Playlist rate, i.e., the percentage of recommended songs added to playlists by users. For confidentiality reasons, we do not report exact Add-To-Playlist rates in this paper, nor the number of users involved in each cohort. Instead, Figure 6 presents *relative* Add-To-Playlist rates with respect to Reference. On average, users exposed to MF-Transformer added 70% more recommended songs to their playlists than those in the reference cohort.

Following this conclusive test, MF-Transformer has been deployed to all users. Future experiments will evaluate other APC models on the service, such as FM-Transformer and NN-Transformer. Additionally, we plan to study the application of our framework to related sequential tasks on Deezer, including radio personalization.

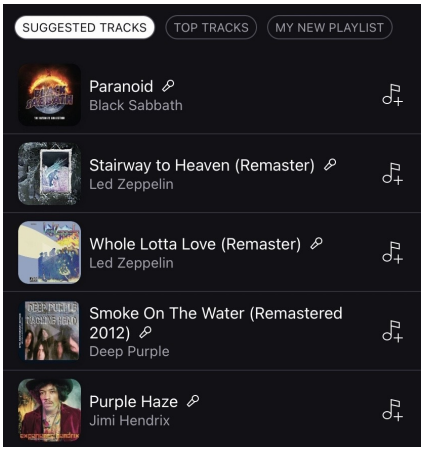


Figure 5: APC on Deezer: songs recommended by our MF-Transformer to continue the playlist from Figure 1.

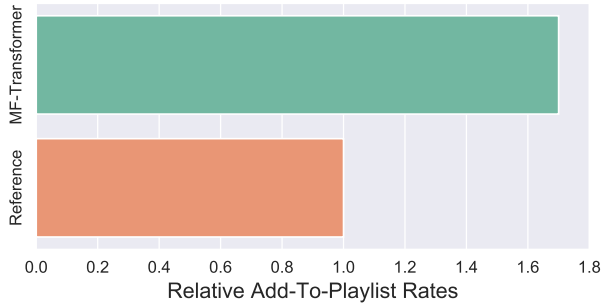


Figure 6: Online A/B test: relative Add-To-Playlist rates compared to the reference model from Deezer. Differences are statistically significant at the 1% level (p-value < 0.01).

Lastly, as recent work [29] identified biases in playlist datasets that can be reproduced by recommender systems, we intend to further study these important aspects in the context of our framework.

6 CONCLUSION

In this paper, we introduced a general framework to build scalable APC models meeting the expected requirements associated with large-scale industrial applications, e.g., on music streaming services. We provided a detailed overview of its possibilities and limitations, showing its versatility in incorporating a wide range of advanced representation learning and sequence modeling techniques, often overlooked in previous large-scale APC experiments due to their complexity. We demonstrated the empirical relevance of our framework through both offline experiments on the largest public dataset for APC, and online experiments, improving APC at scale on a global music streaming service. Besides the already discussed future tests on this same service, our future work will consider multi-modal song representation learning models (e.g., processing audio signals and lyrics in addition to usage data) to enhance the representation part of our framework. We will also aim to incorporate the ability for it to adapt to user feedback in real-time.

REFERENCES

[1] Takuya Akiba, Shotaro Sano, Toshihiko Yanase, Takeru Ohta, and Masanori Koyama. 2019. Optuna: A Next-Generation Hyperparameter Optimization Framework. *Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining* (2019), 2623–2631.

[2] Walid Bendada, Guillaume Salha, and Théo Bontempelli. 2020. Carousel Personalization in Music Streaming Apps with Contextual Bandits. *Proceedings of the Fourteenth ACM Conference on Recommender Systems* (2020), 420–425.

[3] Jesús Bobadilla, Fernando Ortega, Antonio Hernando, and Abraham Gutiérrez. 2013. Recommender Systems Survey. *Knowledge-Based Sys.* 46 (2013), 109–132.

[4] Geoffray Bonnin and Dietmar Jannach. 2014. Automated Generation of Music Playlists: Survey and Experiments. *Comput. Surveys* 47, 2 (2014), 1–35.

[5] Théo Bontempelli, Benjamin Chapus, François Rigaud, Mathieu Morlon, Marin Lorant, and Guillaume Salha-Galvan. 2022. Flow Moods: Recommending Music by Moods on Deezer. *Proceedings of the 16th ACM Conference on Recommender Systems* (2022), 452–455.

[6] Léa Briand, Guillaume Salha-Galvan, Walid Bendada, Mathieu Morlon, and Viet-Anh Tran. 2021. A Semi-Personalized System for User Cold Start Recommendation on Music Streaming Apps. *Proceedings of the 27th ACM SIGKDD Conference on Knowledge Discovery & Data Mining* (2021), 2601–2609.

[7] Ching-Wei Chen, Paul Lamere, Markus Schedl, and Hamed Zamani. 2018. Recsys Challenge 2018: Automatic Music Playlist Continuation. *Proceedings of the 12th ACM Conference on Recommender Systems* (2018), 527–528.

[8] Tran Khanh Dang, Quang Phu Nguyen, and Van Sinh Nguyen. 2020. A Study of Deep Learning-Based Approaches for Session-Based Recommendation Systems. *SN Computer Science* 1, 4 (2020), 1–13.

[9] Yann N Dauphin, Angela Fan, Michael Auli, and David Grangier. 2017. Language Modeling with Gated Convolutional Networks. *Proceedings of the 34th International Conference on Machine Learning* (2017), 933–941.

[10] Tim Donkers, Benedikt Loepp, and Jürgen Ziegler. 2017. Sequential User-Based Recurrent Neural Network Recommendations. *Proceedings of the 11th ACM Conference on Recommender Systems* (2017), 152–160.

[11] Guglielmo Faggioli, Mirko Polato, and Fabio Aiolli. 2018. Efficient Similarity Based Methods for the Playlist Continuation Task. *Proceedings of the ACM Recommender Systems Challenge 2018* (2018), 1–6.

[12] Maurizio Ferrari Dacrema, Paolo Cremonesi, and Dietmar Jannach. 2019. Are We Really Making Much Progress? A Worrying Analysis of Recent Neural Recommendation Approaches. *Proceedings of the 13th ACM Conference on Recommender Systems* (2019), 101–109.

[13] Andres Ferraro, Dmitry Bogdanov, Jisang Yoon, KwangSeob Kim, and Xavier Serra. 2018. Automatic Playlist Continuation using a Hybrid Recommender System Combining Features from Text and Audio. *Proceedings of the ACM Recommender Systems Challenge 2018* (2018), 1–5.

[14] Andres Ferraro, Yuntae Kim, Soohyeon Lee, Biho Kim, Namjun Jo, Semi Lim, Suyon Lim, Jungtaek Jang, Sehwan Kim, Xavier Serra, et al. 2021. Melon Playlist Dataset: A Public Dataset for Audio-Based Playlist Generation and Music Tagging. *Proceedings of the 2021 IEEE International Conference on Acoustics, Speech and Signal Processing* (2021), 536–540.

[15] Diksha Garg, Priyanka Gupta, Pankaj Malhotra, Lovekesh Vig, and Gautam Shroff. 2019. Sequence and Time Aware Neighborhood for Session-Based Recommendations: STAN. *Proceedings of the 42nd International ACM SIGIR Conference on Research and Development in Information Retrieval* (2019), 1069–1072.

[16] Yoav Goldberg and Omer Levy. 2014. word2vec Explained: Deriving Mikolov et al.'s Negative-Sampling Word-Embedding Method. *arXiv:1402.3722* (2014).

[17] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. 2016. *Deep Learning*. MIT Press.

[18] Jyotirmoy Gope and Sanjay Kumar Jain. 2017. A Survey on Solving Cold Start Problem in Recommender Systems. *Proceedings of the 2017 International Conference on Computing, Communication and Automation* (2017), 133–138.

[19] Lei Guo, Hongzhi Yin, Qinyong Wang, Tong Chen, Alexander Zhou, and Nguyen Quoc Viet Hung. 2019. Streaming Session-Based Recommendation. *Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining* (2019), 1569–1577.

[20] Negar Hariri, Bamshad Mobasher, and Robin Burke. 2015. Adapting to User Preference Changes in Interactive Recommendation. *Proceedings of the 24th International Joint Conference on Artificial Intelligence* (2015).

[21] Xiangnan He, Lizi Liao, Hanwang Zhang, Liqiang Nie, Xia Hu, and Tat-Seng Chua. 2017. Neural Collaborative Filtering. *Proceedings of the 26th International Conference on World Wide Web* (2017), 173–182.

[22] Balázs Hidasi, Alexandros Karatzoglou, Linas Baltrunas, and Domonkos Tikk. 2016. Session-Based Recommendations with Recurrent Neural Networks. *Proceedings of the 4th International Conference on Learning Representations* (2016).

[23] Yifan Hu, Yehuda Koren, and Chris Volinsky. 2008. Collaborative Filtering for Implicit Feedback Datasets. *Proceedings of the 8th IEEE International Conference on Data Mining* (2008), 263–272.

[24] Kurt Jacobson, Vidhya Murali, Edward Newett, Brian Whitman, and Romain Yon. 2016. Music Personalization at Spotify. *Proceedings of the 10th ACM Conference on Recommender Systems* (2016), 373–373.

[25] Laurie Jakobsen. 2016. Playlists Overtake Albums in Listenership, says Loop Study. *Blog post on the "Music Business Association": https://musicbiz.org/news/playlists-overtake-albums-listenership-says-loop-study/* (2016).

[26] Dietmar Jannach, Gabriel de Souza P. Moreira, and Even Oldridge. 2020. Why Are Deep Learning Models Not Consistently Winning Recommender Systems Competitions Yet? A Position Paper. *Proceedings of the Recommender Systems Challenge 2020* (2020), 44–49.

[27] How Jing and Alexander J Smola. 2017. Neural Survival Recommender. *Proceedings of the 10th ACM International Conference on Web Search and Data Mining* (2017), 515–524.

[28] Wang-Cheng Kang and Julian McAuley. 2018. Self-Attentive Sequential Recommendation. *Proceedings of the 2018 IEEE International Conference on Data Mining* (2018), 197–206.

[29] Peter Knees, Andres Ferraro, and Moritz Hübler. 2022. Bias and Feedback Loops in Music Recommendation: Studies on Record Label Impact. In *Workshop of Multi-Objective Recommender Systems (MORS'22), in conjunction with the 16th ACM Conference on Recommender Systems*, Vol. 22. 2022.

[30] Yehuda Koren and Robert Bell. 2015. Advances in Collaborative Filtering. *Recommender Systems Handbook* (2015), 77–118.

[31] Yehuda Koren, Robert Bell, and Chris Volinsky. 2009. Matrix Factorization Techniques for Recommender Systems. *Computer* 42, 8 (2009), 30–37.

[32] Dominik Kowald, Markus Schedl, and Elisabeth Lex. 2020. The Unfairness of Popularity Bias in Music Recommendation: A Reproducibility Study. *Proceedings of the 42nd European Conference on Information Retrieval* (2020), 35–42.

[33] Malte Ludewig and Dietmar Jannach. 2018. Evaluation of Session-Based Recommendation Algorithms. *User Modeling and User-Adapted Interaction* 28, 4 (2018), 331–390.

[34] Malte Ludewig, Iman Kamehkhosh, Nick Landia, and Dietmar Jannach. 2018. Effective Nearest-Neighbor Music Recommendations. *Proceedings of the ACM Recommender Systems Challenge 2018* (2018), 1–6.

[35] Malte Ludewig, Noemi Mauro, Sara Latifi, and Dietmar Jannach. 2019. Performance Comparison of Neural and Non-Neural Approaches to Session-Based Recommendation. *Proceedings of the 13th ACM Conference on Recommender Systems* (2019), 462–466.

[36] Malte Ludewig, Noemi Mauro, Sara Latifi, and Dietmar Jannach. 2021. Empirical Analysis of Session-Based Recommendation Algorithms. *User Modeling and User-Adapted Interaction* 31, 1 (2021), 149–181.

[37] Brian McFee and Gert Lanckriet. 2012. Hypergraph Models of Playlist Dialects. *Proceedings of the 13th International Society for Music Information Retrieval Conference* (2012), 343–348.

[38] Ruihui Mu. 2018. A Survey of Recommender Systems Based on Deep Learning. *IEEE Access* 6 (2018), 69009–69022.

[39] Alec Radford, Karthik Narasimhan, Tim Salimans, Ilya Sutskever, et al. 2018. Improving Language Understanding by Generative Pre-Training. *OpenAI* (2018).

[40] Steffen Rendle. 2010. Factorization Machines. *Proceedings of the 2010 IEEE International Conference on Data Mining* (2010), 995–1000.

[41] Steffen Rendle, Walid Krichene, Li Zhang, and John Anderson. 2020. Neural Collaborative Filtering vs. Matrix Factorization Revisited. *Proceedings of the Fourteenth ACM Conference on Recommender Systems* (2020), 240–248.

[42] Francesco Ricci, Lior Rokach, and Bracha Shapira. 2011. Introduction to Recommender Systems Handbook. *Recommender Systems Handbook* (2011), 1–35.

[43] Vasiliy Rubtsov, Mikhail Kamenshchikov, Ilya Valyaev, Vasiliy Leksin, and Dmitry I Ignatov. 2018. A Hybrid Two-Stage Recommender System for Automatic Playlist Continuation. *Proceedings of the ACM Recommender Systems Challenge 2018* (2018), 1–4.

[44] Guillaume Salha-Galvan, Romain Hennequin, Benjamin Chapus, Viet-Anh Tran, and Michalis Vazirgiannis. 2021. Cold Start Similar Artists Ranking with Gravity-Inspired Graph Autoencoders. *15th ACM Conference on Recommender Systems* (2021), 443–452.

[45] Markus Schedl. 2019. Deep Learning in Music Recommendation Systems. *Frontiers in Applied Mathematics and Statistics* (2019), 44.

[46] Markus Schedl, Hamed Zamani, Ching-Wei Chen, Yashar Deldjoo, and Mehdi Elahi. 2018. Current Challenges and Visions in Music Recommender Systems Research. *Int. Journal of Multimedia Information Retrieval* 7, 2 (2018), 95–116.

[47] Weiping Song, Chence Shi, Zhiping Xiao, Zhijian Duan, Yewen Xu, Ming Zhang, and Jian Tang. 2019. AutoInt: Automatic Feature Interaction Learning via Self-Attentive Neural Networks. *Proceedings of the 28th ACM International Conference on Information and Knowledge Management* (2019), 1161–1170.

[48] Nitish Srivastava, Geoffrey Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. 2014. Dropout: a Simple Way to Prevent Neural Networks from Overfitting. *The Journal of Machine Learning Research* 15, 1 (2014), 1929–1958.

[49] Xiaoyuan Su and Taghi M Khoshgoftaar. 2009. A Survey of Collaborative Filtering Techniques. *Advances in Artificial Intelligence* 2009 (2009).

[50] Fei Sun, Jun Liu, Jian Wu, Changhua Pei, Xiao Lin, Wenwu Ou, and Peng Jiang. 2019. BERT4Rec: Sequential Recommendation with Bidirectional Encoder Representations from Transformer. *Proceedings of the 28th ACM International Conference on Information and Knowledge Management* (2019), 1441–1450.

[51] Roberto Turrin, Massimo Quadrana, Andrea Condorelli, Roberto Pagano, and Paolo Cremonesi. 2015. 30Music Listening and Playlists Dataset. *RecSys Posters* (2015), 75.

[52] Flavian Vasile, Elena Smirnova, and Alexis Conneau. 2016. Meta-Prod2vec: Product Embeddings using Side-Information for Recommendation. *Proceedings of the 10th ACM conference on recommender systems* (2016), 225–232.

[53] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. 2017. Attention Is All You Need. *Advances in Neural Information Processing Systems* 30 (2017).

[54] Maksims Volkovs, Himanshu Rai, Zhaoyue Cheng, Ga Wu, Yichao Lu, and Scott Sanner. 2018. Two-Stage Model for Automatic Playlist Continuation at Scale. *Proceedings of the ACM Recommender Systems Challenge 2018* (2018), 1–6.

[55] Dongjing Wang, Shuiguang Deng, Xin Zhang, and Guandong Xu. 2016. Learning Music Embedding with Metadata for Context Aware Recommendation. *Proceedings of the 2016 ACM on International Conference on Multimedia Retrieval* (2016), 249–253.

[56] Shoujin Wang, Liang Hu, Yan Wang, Longbing Cao, Quan Z Sheng, and Mehmet Orgun. 2019. Sequential Recommender Systems: Challenges, Progress and Prospects. *Proceedings of the 28th International Joint Conference on Artificial Intelligence* (2019).

[57] Yining Wang, Liwei Wang, Yuanzhi Li, Di He, Wei Chen, and Tie-Yan Liu. 2013. A Theoretical Analysis of NDCG Ranking Measures. In *26th Annual Conference on Learning Theory*, Vol. 8. 6.

[58] Hojin Yang, Yoonki Jeong, Minjin Choi, and Jongwuk Lee. 2018. MMCF: Multimodal Collaborative Filtering for Automatic Playlist Continuation. *Proceedings of the ACM Recommender Systems Challenge 2018* (2018), 1–6.

[59] Fajie Yuan, Alexandros Karatzoglou, Ioannis Arapakis, Joemon M Jose, and Xiangnan He. 2019. A Simple Convolutional Generative Network for Next Item Recommendation. *Proceedings of the 12th ACM International Conference on Web Search and Data Mining* (2019), 582–590.

[60] Hamed Zamani, Markus Schedl, Paul Lamere, and Ching-Wei Chen. 2019. An Analysis of Approaches Taken in the ACM RecSys Challenge 2018 for Automatic Music Playlist Continuation. *ACM Transactions on Intelligent Systems and Technology (TIST)* 10, 5 (2019), 1–21.

[61] Eva Zangerle, Martin Pichl, Wolfgang Gassler, and Günther Specht. 2014. #nowplaying Music Dataset: Extracting Listening Behavior from Twitter. *Proceedings of the 1st Int. Workshop on Internet-Scale Multimedia Management* (2014), 21–26.

[62] Shuai Zhang, Lina Yao, Aixin Sun, and Yi Tay. 2019. Deep Learning Based Recommender System: A Survey and New Perspectives. *Comput. Surveys* 52, 1 (2019), 1–38.